Lasso & Ridget Regressions

Luis Henrique

2024-09-02

`{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE)`

Lasso e Ridget Penalty for Regressions models

Quando pensamos em regressões lineares, o uso mais comum e conceitual é para previsões de acontecimentos futuros. Nessa abordagem, o objetivo principal é encontrar a função $g(x)$ que minimize a função de risco $R(g(x))$ e que generalize bem para novos dados.

Contudo, enfrentamos um problema conceitual com a MSE (Mean Squared Error), que é a função de custo mais utilizada. A MSE tende a reduzir-se quando a complexidade do modelo aumenta, o que pode levar a um problema de *overfitting*. Overfitting ocorre quando o modelo se ajusta excessivamente aos dados de treinamento, perdendo a capacidade de generalizar para novos dados.

Para enfrentar o overfitting, aplicamos técnicas como *Data Splitting* e *Cross Validation*. Essas abordagens ajudam a avaliar o modelo de forma mais robusta, separando os dados em conjuntos de treinamento e teste e validando o modelo em diferentes subconjuntos dos dados. No entanto, além dessas técnicas, é importante lidar com o trade-off entre viés e variância.

O trade-off entre viés e variância refere-se à troca entre a complexidade do modelo e sua capacidade de generalização. Modelos mais complexos podem ter baixa variância (ou seja, ajustam-se muito bem aos dados de treinamento) mas alto viés (ou seja, não generalizam bem para novos dados). Para mitigar isso, aplicamos penalidades para reduzir a complexidade do modelo e melhorar sua capacidade de generalização.

Quando lidamos com conjuntos de dados de alta dimensionalidade, ou seja, com muitas variáveis de entrada, a complexidade dos modelos pode causar overfitting devido à grande quantidade de features. Nesse contexto, adicionamos parâmetros de ajuste, conhecidos como penalidades, às funções de regressão. As duas principais penalidades são Lasso (L1) e Ridge (L2).

Penalidade Lasso (L1)

A penalidade Lasso adiciona um termo de regularização à função de custo que penaliza a soma dos valores absolutos dos coeficientes:

$$\text{Lasso } R(g(x)) = \text{MSE} + \lambda \sum_{j=1}^{p} |\beta_j|$$

Onde λ é o parâmetro de regularização que controla a intensidade da penalidade. O efeito principal da penalidade Lasso é forçar alguns coeficientes a serem exatamente zero, efetivamente realizando seleção de variáveis e reduzindo a complexidade do modelo.

Penalidade Ridge (L2)

A penalidade Ridge adiciona um termo de regularização que penaliza a soma dos quadrados dos coeficientes:

$$\text{Ridge } R(g(x)) = \text{MSE} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Aqui, λ também controla a intensidade da penalidade, mas o Ridge tende a reduzir todos os coeficientes de forma mais uniforme, sem forçá-los a zero. Isso ajuda a reduzir a variância do modelo sem eliminar completamente variáveis.

Ambas as técnicas ajudam a controlar a complexidade do modelo e a melhorar a capacidade de generalização ao lidar com dados de alta dimensionalidade. A escolha entre Lasso e Ridge depende das características do problema e dos objetivos do modelo.

Vamos ver como isso funciona na prática

Seguindo o exemplo do livro "Aprendizado de Máquina - Uma abordagem estatística" de Rafael Izbicki (que inclusive recomendo a leitura) ele trabalha, ***no cap. 3 parte 3.8 no exemplo 3.3*** com dados da Amazon Fine Food Reviews do link https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

O código começa carregando um conjunto de dados de avaliações a partir de um arquivo CSV e seleciona uma amostra aleatória de 20.000 linhas para análise, garantindo a reprodutibilidade ao definir uma semente para a amostragem. Em seguida, ele cria um corpus de texto a partir da coluna de texto das avaliações e constrói uma matriz de documentos e termos (DTM) que representa a frequência das palavras em cada documento. Essa matriz é então convertida em uma forma esparsa para otimizar o armazenamento, considerando que muitas entradas são zero.

Após a preparação dos dados, o código divide aleatoriamente os dados em conjuntos de treinamento e teste, selecionando 10.000 índices para treinamento. Em relação ao tratamento de dados, o código calcula a média dos escores de avaliação, ignorando os valores ausentes (NAs), e substitui os NAs nos escores de teste por essa média. Esse tratamento é essencial para evitar problemas durante a validação do modelo, assegurando que o modelo possa ser avaliado de forma eficaz e sem distorções causadas por valores ausentes.

Obs: Lembrando que essa parte é dos créditos do professor Rafael, eu simplesmente fiz um CTRL-C e CTRL-V

```{r cars} # Carregar as bibliotecas necessárias library(dplyr) library(tm) library(corrplot) library(car) library(ggplot2) library(glmnet) library(patchwork) library(tidyr) library(reshape2) library(gridExtra)

## Ler os dados da amostra

dados <- read.csv('Reviews.csv', header = TRUE)

## Selecionar uma amostra dos dados

set.seed(1) selecao <- sample(nrow(dados), 20000) #selecionando uma amostra dos dados dados <- dados[selecao,]

## Criar a matriz de preditoras (Document-Term Matrix)

corp <- VCorpus(VectorSource(dados
$Text¿¿ dtm <- DocumentTermMatrix(corp, control=list(tolower=TRUE, stemming=FAL$
i, j = dtm$j, x=dtm$v, dimnames = list(NULL, dtm$dimnames[[2]]) ¿, dims=c ¿$nrow, dtm$ncol)) dim(dtmMatrix)

tr <- sample.int(length(selecao), 10000, replace = FALSE)

## Calcular a média ignorando os NAs -> Esses valores atrapalham validações

media_score <- mean(dados$Score[-tr], na.rm = TRUE)

## Substituir os NAs pela média calculada

dados$Score[-tr] ¿$Score[-tr])] <- media_score

```
Veja a matriz criada, tendo mais de 35mil features (termos) e 20mil
instâncias

Aqui $d > n$ mostra a imensa dimensionalidade causada de forma
proposital.

Criação de Modelos de Regressão

Neste segmento, o autor desenvolve três modelos de regressão: um
modelo baseado em Mínimos Quadrados Ordinários (MQO), um modelo de
Ridge e um modelo de Lasso.
```
```

Primeiramente, é criado um modelo de regressão linear simples utilizando MQO. Em seguida, o autor adapta o modelo para incorporar penalidades Ridge e Lasso. A aplicação da penalidade Ridge envolve a adição de um termo de regularização baseado na soma dos quadrados dos coeficientes, enquanto a penalidade Lasso adiciona um termo de regularização baseado na soma dos valores absolutos dos coeficientes.

Após a construção dos modelos, realiza-se a previsão para cada um deles. É importante notar que o pacote `glmnet` utiliza validação cruzada para determinar automaticamente o valor ideal do parâmetro de ajuste, λ. No entanto, em seções subsequentes, abordaremos a metodologia para selecionar esse parâmetro de forma manual, permitindo uma compreensão mais profunda do processo de ajuste dos modelos

````
```{r pressure}
# Ajuste dos modelos de regressão

# Modelo MQO (Mínimos Quadrados Ordinários)
ajuste_mq <- glmnet(dtmMatrix[tr,], dados$Score[tr], alpha = 0, lambda
= 0)
predito_mq <- predict(ajuste_mq, newx = dtmMatrix[-tr,])

# Modelo Ridge
ridge <- cv.glmnet(dtmMatrix[tr,], dados$Score[tr], alpha = 0)
predito_ridge <- predict(ridge, s = ridge$lambda.min, newx =
dtmMatrix[-tr,])

# Modelo Lasso
lasso <- cv.glmnet(dtmMatrix[tr,], dados$Score[tr], alpha = 1)
predito_lasso <- predict(lasso, s = lasso$lambda.min, newx =
dtmMatrix[-tr,])
````

Agora vamos montar uma tabela cruzando dados previstos com dados reais para encontrarmos os resultados resíduais de todos os três modelos.

A partir daqui eu apenas me inspirei no livro, porém os códigos são de autoria própria

```
# Função para calcular o EQM e o erro padrão com tratamento de NA
calcular_metrica_eficiente <- function(predito, real) {
  # Remover valores NA
  predito <- na.omit(predito)
  real <- na.omit(real)

  # Verificar se os vetores têm o mesmo comprimento
  if (length(predito) != length(real)) {
    stop("Os vetores 'predito' e 'real' têm comprimentos diferentes.")
  }

  # Calcular o erro e as métricas
```
```

```
  erro <- (predito - real)^2
  eqm <- mean(erro)
  erro_padrao <- sd(erro) / sqrt(length(erro))
  return(c(eqm, erro_padrao))
}

# Calcular métricas para cada modelo
metrica_mqo <- calcular_metrica_eficiente(as.vector(predito_mq),
dados$Score[-tr])
metrica_lasso <- calcular_metrica_eficiente(as.vector(predito_lasso),
dados$Score[-tr])
metrica_ridge <- calcular_metrica_eficiente(as.vector(predito_ridge),
dados$Score[-tr])

# Criar a tabela de métricas
tabela_metricas <- data.frame(
  Modelo = c("MQO", "Lasso", "Ridge"),
  EQM = c(metrica_mqo[1], metrica_lasso[1], metrica_ridge[1]),
  Erro_Padrao = c(metrica_mqo[2], metrica_lasso[2], metrica_ridge[2])
)

tabela_metricas
```

Desempenho do Modelo Lasso

O modelo Lasso demonstrou ser o mais eficaz entre os três analisados. O Lasso aplica uma penalidade à função de custo que é proporcional à soma das magnitudes dos coeficientes multiplicados pelo parâmetro de ajuste λ. Esta abordagem é particularmente vantajosa em cenários de alta dimensionalidade, pois tem a capacidade de eliminar features irrelevantes de maneira mais assertiva.

Embora o modelo Ridge também tenha mostrado um desempenho próximo, o Lasso apresentou uma vantagem significativa ao reduzir mais efetivamente o número de features. No entanto, essa eliminação excessiva de features pode, às vezes, impactar negativamente a capacidade de generalização do modelo para novos dados, já que o Lasso tende a promover um maior número de coeficientes zero comparado ao Ridge.

Portanto, enquanto o Lasso é eficaz na seleção de features e na redução da dimensionalidade, é crucial balancear essa seleção com a necessidade de manter a capacidade preditiva do modelo.

```
# Transformar a tabela em formato longo
tabela_metricas_long <- melt(tabela_metricas, id.vars = "Modelo",
variable.name = "Métrica", value.name = "Valor")

# Garantir que os modelos apareçam na ordem desejada
tabela_metricas_long$Modelo <- factor(tabela_metricas_long$Modelo,
levels = c("MQO", "Lasso", "Ridge"))

# Plotar o gráfico de barras com valores
```

```
ggplot(tabela_metricas_long, aes(x = Modelo, y = Valor, fill =
Métrica)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(Valor, 2)),
            position = position_dodge(width = 0.9),
            vjust = -0.5, size = 4) +
  labs(title = "Comparação de Métricas dos Modelos", x = "Modelo", y =
"Valor") +
  scale_fill_manual(values = c("EQM" = "cornflowerblue", "Erro_Padrao"
= "tomato2")) +
  theme_classic()
```

Vamos dar uma breve olhada para quais palavras apresentam maior importância para a previsão no nosso modelo Lasso. Quanto maior o coeficiente melhor e mais importante ele se torna à previsão

```
# Obtenha os coeficientes do Lasso e remova o intercepto
coeficientes_lasso <- as.matrix(coef(lasso, s = lasso$lambda.min))
coeficientes_lasso <- coeficientes_lasso[-1, , drop = FALSE]  #
Remover o intercepto

# Converter em dataframe e adicionar os nomes das palavras
coeficientes_lasso_df <- data.frame(
  Palavra = rownames(coeficientes_lasso),
  Coeficiente = coeficientes_lasso[, 1]
)

# Separar em positivos e negativos
coeficientes_positivos <- coeficientes_lasso_df %>%
  filter(Coeficiente > 0) %>%
  arrange(desc(Coeficiente))

coeficientes_negativos <- coeficientes_lasso_df %>%
  filter(Coeficiente < 0) %>%
  arrange(Coeficiente)

# Gerar gráfico para coeficientes positivos
grafico_positivos <- ggplot(coeficientes_positivos[1:20, ], aes(x =
Coeficiente, y = reorder(Palavra, Coeficiente))) +
  geom_col(fill = "cornflowerblue") +
  labs(x = "Coeficiente Lasso", title = "Coeficientes Positivos") +
  theme_classic()

# Gerar gráfico para coeficientes negativos
grafico_negativos <- ggplot(coeficientes_negativos[1:20, ], aes(x =
Coeficiente, y = reorder(Palavra, Coeficiente))) +
  geom_col(fill = "tomato") +
  labs(x = "Coeficiente Lasso", title = "Coeficientes Negativos") +
  theme_classic()
```

```
# Colocar os gráficos lado a lado
grafico_positivos + grafico_negativos
```

Veja como temos boas features mas a partir da décima quinta ou vigésima feature seu coef se aproxima de 0. Aqui mostramos apenas 20, mas lembre-se que são muito mais de 20, veja que quanto mais features acrescentamos, menos importância algumas tem, considerar modelos de regressão com as mesmas infla e gera overfitting no modelo piorando sua previsão.

Poderíamos considerar isso em regressões inferenciais, visto que a ideia não é predizer valores futuros mas avaliar uma regressão “real” do modelo e verificar o cruzamento das features com seus rótulos.

Por fim, como prometido, vou te mostrar um pouco de como selecionar bons valores de tuning params ou o lambda

Para isso, vamos criar uma lista de diversos valores de lambda e calcular, para cada, a função de risco. Desses resultados, vamos plotar uma curva para representar os resultados.

```
# Definir a sequência de valores de lambda
lambda_seq <- seq(0, 1, by = 0.1)

# Ajustar o modelo com glmnet para cada valor de lambda
lasso_model <- glmnet(dtmMatrix[tr,], dados$Score[tr], alpha = 1,
lambda = lambda_seq)

# Realizar a validação cruzada e calcular o erro de previsão e
penalidade
mse_penalidade <- numeric(length(lambda_seq))

for (i in 1:length(lambda_seq)) {
  # Prever usando o valor de lambda
  predito <- predict(lasso_model, s = lambda_seq[i], newx =
dtmMatrix[-tr,])

  # Calcular o erro de previsão (MSE)
  erro <- mean((dados$Score[-tr] - predito)^2)

  # Calcular a penalidade
  penalidade <- lambda_seq[i] * sum(coef(lasso_model, s =
lambda_seq[i])[-1] != 0)

  # Calcular a função de risco (MSE + Penalidade)
  mse_penalidade[i] <- erro + penalidade
}

# Criar um data frame para plotagem
resultados <- data.frame(
  Lambda = lambda_seq,
```

```
  Funcao_Risco = mse_penalidade
)

# Encontrar o valor mínimo da função de risco e o correspondente
lambda
indice_min <- which.min(resultados$Funcao_Risco)
lambda_min <- resultados$Lambda[indice_min]
risco_min <- resultados$Funcao_Risco[indice_min]

# Plotar o gráfico com linha azul e suavização
ggplot(resultados, aes(x = Lambda, y = Funcao_Risco)) +
  geom_line(color = "lightblue", size = 0.8) +  # Linha azul
  geom_vline(xintercept = lambda_min, linetype = "dashed", color =
"tomato") +  # Linha vertical tracejada
  geom_text(aes(x = lambda_min+0.15, y = risco_min, label =
sprintf("Lambda = %.1f", lambda_min)),
            vjust = -0.5, hjust = 1.1, color = "tomato", size = 3) +
# Rótulo
  labs(x = "Lambda", y = "Função de Risco (MSE + Penalidade (P0))",
       title = "Função de Risco vs Lambda para Lasso") +
  theme_classic()
```

Sendo que o Lasso utilizado pelo glmnet foi:

```
# Obtém o lambda que minimiza o erro de validação cruzada
lambda_min <- lasso$lambda.min

# Obtém o lambda com o menor erro de validação cruzada mais um desvio
padrão
lambda_1se <- lasso$lambda.1se

# Imprime os valores de lambda
cat("Lambda que minimiza o erro de validação cruzada:", lambda_min, "\
n")
cat("Lambda com o menor erro de validação cruzada mais um desvio
padrão:", lambda_1se, "\n")
```

Assim, podemos ver o quão benéfico é as regularizações de Lasso e Ridget para datasets de alta dimensão.

Comumente é recomendado o uso da Regressão de Ridge, porém caso acredite que tenham features que não são importantes, a Lasso ou Elastic Net (uma soma das regularizações de Lasso e Ridget) são as mais recomendadas, sendo a Elastic Net mais recomendado.

/————————————————————————————————————-/

Para um bônus vou te mostrar uma forma interessante de plotar esses gráficos de Tuning x MSE

Basta usar o plot(modelo_lasso ou modelo_ridge). Veja que legal:

```
plot1 <- plot(lasso)
plot2 <- plot(ridge)

grid.arrange(plot1, plot2, ncol=2)
```